**Project Design Phase**

**Solution Architecture**

|  |  |
| --- | --- |
| Date | 15 JUNE 2025 |
| Team ID | LTVIP2025TMID46308 |
| Project Name | DOC SPOT |
| Maximum Marks | 4 Marks |

**Solution Architecture:**

The architecture behind **DocSpot** is thoughtfully designed to solve a real-world healthcare issue: the delay and difficulty in booking doctor appointments. Rather than just building an app, DocSpot takes a structured approach that combines usability, performance, and security to ensure a **complete solution for patients, doctors, and administrators**.

The system is built using a **layered architecture** where each component plays a vital role in delivering a smooth, scalable, and secure experience.

**Objectives of the Architecture**

The architecture of DocSpot is guided by these key goals:

* Make doctor appointments easy and quick for patients.
* Help doctors organize their schedule and appointments.
* Allow admins to manage doctor approvals and system health.
* Ensure all users have secure, role-based access.
* Be flexible and scalable to support future upgrades (like video calls or digital prescriptions).

**Architectural Layers & Components**

**1. User Interface (Frontend Layer)**

* **Tech Used**: React.js with Material UI / Bootstrap
* **Purpose**: This is the face of DocSpot — what users interact with directly.
* **How it works**:
  + Patients can sign up, search for doctors, and book appointments.
  + Doctors can manage their availability and respond to bookings.
  + Admins can log in to manage users and oversee the platform.

**2. Server Logic (Backend Layer)**

* **Tech Used**: Node.js with Express.js
* **Purpose**: This is where all the behind-the-scenes logic happens.
* **What it does**:
  + Handles requests from the frontend (e.g., booking, login).
  + Applies business logic like validating appointments or approving doctors.
  + Secures routes and ensures only authorized users access certain features.

**3. Data Storage (Database Layer)**

* **Tech Used**: MongoDB with Mongoose
* **Purpose**: The system’s memory — it stores all important information.
* **Stores**:
  + User profiles (patients, doctors, admins)
  + Appointment data and schedules
  + Doctor availability and admin approval status
* **All data is schema-validated and stored securely.**

**4. User Security (Authentication & Authorization)**

* **Tech Used**: JSON Web Tokens (JWT)
* **Purpose**: Keeps the platform secure and ensures proper access control.
* **Key Roles**:
  + Authenticates users at login.
  + Provides secure access tokens for API communication.
  + Restricts features based on user roles (patient/doctor/admin).

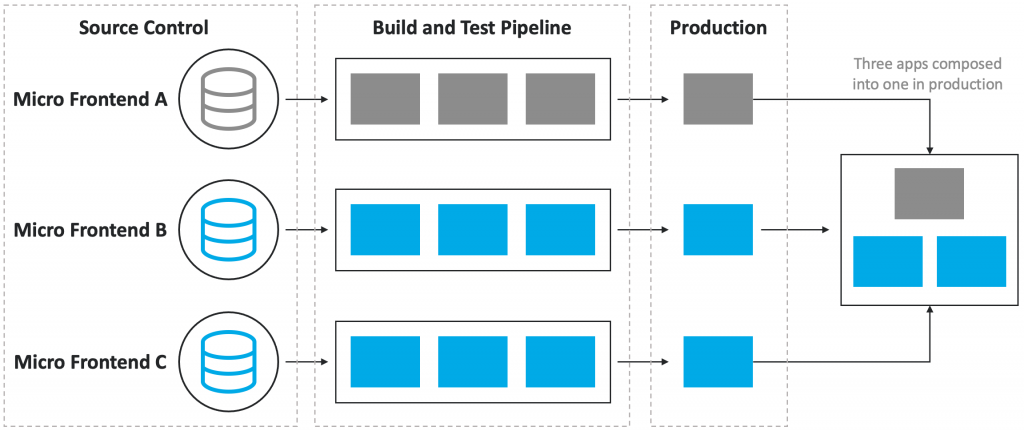
**5. Admin Dashboard (Control Panel)**

* **Purpose**: Provides oversight and control over the system.
* **Features**:
  + View all users and appointments.
  + Approve or reject doctor registrations.
  + Block/unblock users as needed.
  + Ensure trust and transparency within the platform.

**6. Hosting & Deployment (Infrastructure Layer)**

* **Technologies**:
  + **Frontend**: Hosted on Vercel or Netlify
  + **Backend**: Deployed using Render or Railway
  + **Database**: MongoDB Atlas (cloud-based)
* **Why this matters**:
  + Ensures the application is live, fast, and reliable.
  + Makes scaling easier as the user base grows.
  + Supports continuous deployment and testing.

**Example - Solution Architecture Diagram:**



*Figure 1: Architecture and data flow of the seamless appointment booking*